Exploring Linear Feature Scalability of Vision Transformer for Parameter-efficient Fine-tuning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Existing fine-tuning schemes for vision transformers mostly involve updating all model parameters to adapt to a new target task. Such a practice, however, suffers from two main drawbacks: i) the original information preserved within the model is inevitably eliminated, making it unavailable for other target tasks; ii) the source dataset of the pre-training network is typically significantly larger than the target dataset, hence updating all parameters seems to be unnecessary and a waste of computational resources. In this paper, we explore a simple yet effective strategy for adapting pre-trained models, termed as LInear Feature Scalability (LIFTs). Unlike prior approaches that rely on updating the pre-trained weights of the network for downstream adaptation, LIFTs preserves 100% model parameters unaltered from the pre-trained network, thereby making the model readily reusable across a wide range of tasks. Specifically, we introduce a novel task-specific linear feature scalability adaptation layer delicately designed for each block, which takes into account the linear transformation of the features. In this way, we surprisingly show that such a linear transformation is capable of handling efficient fine-tuning. Extensive experiments on downstream tasks show the effectiveness and superior performance of our proposed LIFTs. Code will be available.

1 Introduction

As data-driven learning methods make their way as a promising training technique into deep learning community, the dataset scale and the model size have both got huge explosions (*e.g.*, ViT-G/14 [7] 1.8G, CoAtNet [3] 2.4G). Most researchers have adopted these pre-trained models for downstream tasks to achieve better performance and faster convergence, which gradually becomes a common way.

However, the current procedure depend on full fine-tuning heavily, meaning *all* the parameters of the backbone need fine-tuning. This common practice inevitably leads to two issues: i) after the fine-tuning, the model is over-fitted to the small target dataset and thus cannot be used for other tasks. As a results, each task will require a dedicated set of model parameters in the training stage, which causes a huge amount of storage space; ii) as all model parameters are updated, the fine-tuning process will take a significant amount of computations; since the fine-tuning is needed for all downstream tasks, the total computational cost will be even larger, sometimes to a degree that is beyond the reach of a common user.

A simple solution for the above two problems is linear probe [10], where only the last linear layer is fine-tuned. However, this practice usually yields inferior performance compared to the full fine-tuning proxy. Motivated by the success of the efficient parameter fine-tuning strategy with prompt in the

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

field of natural language processing (NLP) [14, 22, 17, 14, 13], the recent work implement a similar proxy on vision tasks [21], termed as Visual Prompt Tuning (VPT). Specifically, VPT [21] proposes to insert learnable prompts and append them to the input tokens. The concatenated tokens are then used as the input to the model. These prompts will interact with the image tokens with self-attention and are updated during the fine-tuning process, as shown in Figure 1. In this manner, a significant performance improvement can be achieved in downstream tasks compared to a naive probing proxy. Nevertheless, it additionally raises two issues: i) the fine-tuning performance is sensitive to the number of prompts for each task and needs to be carefully designed in VPT. If the number is set too small, the representation ability of the model might not be sufficient, thus degrading the fine-tuned accuracy. On another hand, if the number of prompts is set too large, it will increase redundancy and computational complexity (*e.g.*, 200 prompts on Clevr/count *vs.* 1 prompt on Flowers102). In addition, self-attention makes FLOPs grow quadratically with the number of inserted prompts (*i.e.*, $O(n^2)$), which brings a greater computational cost both during the training and inference stages; ii) such a design that depends on additional inputs and extracts information through self-attention is not a plug and play proxy. For examples, for the class-token-free models [24, 36, 39], inserting the extra prompts is equivalent to training some additional class tokens. As pointed out in [36], this will cause inconvenience when the resolution of the input images changes.

To cope with the above problems, we attempt to find a general proxy for parameter-efficient fine-tuning in vision transformers, where all parameters in the pre-trained model and the number of tokens are kept unchanged. To this end, we revisit some parameter-efficient fine-tuning methods in NLP [15], where they try to insert an adapter into the network and such an adapter with a few trainable parameters will transfer the feature to the source space to target space. However, they employ complex non-linear function (*e.g.*, ReLU) to learn feature mapping. The simple linear transformation is more or less neglected. However, the linear systems usually have better qualities, such as additivity and homogeneity properties. A linear transformation layer is also able to be merged with other components of the network during the inference phase with model re-parameterization [5]. Therefore, to complement this point of the area, we try to explore the effectiveness of the linear adapter.

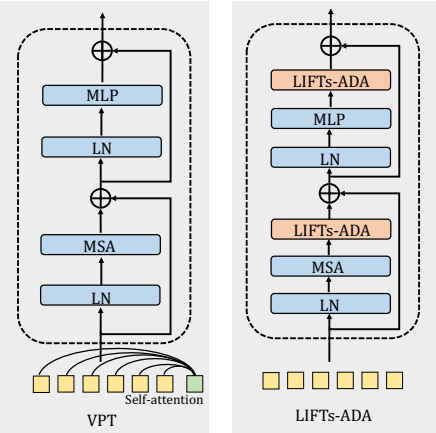


Figure 1: Comparison of architectures between VPT and LIFTs-ADA.

Without loss of generality, we first define *feature scalability*, which describes how good the performance of fine-tuning could be achieved when performing some transformations on this feature while keeping the parameters frozen. If the performance is better, we consider that the feature scalability at this location is stronger. Under this definition, the previous works [15] indeed explore the feature scalability with a non-linear mapping. From another perspective, we explore linear feature scalability with a linear transformation, thus we name our method as LInear FeaTure scalability (LIFTs). Furthermore, we design a linear adapter module, named LIFTs-ADA (LInear FeaTure scalability adapter), to transform the features to the desired distribution of the target dataset. Since our adapter is fully linear, its parameters can be merged into the existing framework modules during the inference stage, thus achieving zero extra parameters and FLOPs for downstream tasks.

Through comprehensive experiments, we surprisingly find that a simple linear adapter (LIFTs-ADA) is capable of handling efficient fine-tuning. It is also plug-and-play and is very easy to extend to a variety of network architectures. Our method, LIFTs, achieves comparable or even better performance with VPT while using fewer parameters and FLOPs in both training and inference stages. We hope that our approach brings more insight into the field of parameter-efficient fine-tuning.

2 Related Work

2.1 Transformers

Convolution has been used for a long time as the main module to extract the image features in computer vision tasks, and CNN-based architectures have been heavily studied [31, 12, 37, 25]. Recently, another training scheme, Transformer, has gained widespread attention owing to its great success in NLP [35, 4, 17]. Following this direction, Dosovitskiy *et al.* [7] first employ a transformer in the domain of computer vision and introduce a new architecture paradigm, ViT, which achieves promising results. Subsequently, various transformer-based models, such as DeiT [33] and Swin Transformer [24], are introduced and shown to be effective on a variety of tasks such as object detection, semantic segmentation, action recognition [26], *etc.*, and these models achieve state-of-the-art results on the relevant datasets. However, in addition to the well-designed modules, their excellent performance is also attributed to the large scales of the models. Given a large-scale model pre-trained on a large dataset, how to perform efficient parameter fine-tuning in downstream tasks is essential but is currently less explored. In this paper, we explore linear feature scalability of vision transformer with comprehensive validation and shows a promising performance.

2.2 Pre-training and Fine-tuning

Early models [12, 18, 16, 37, 32] are usually pre-trained on the ImageNet-1K dataset, and then fine-tuned on downstream tasks to achieve faster convergence [11] or better performance. Such a procedure is called pre-training and fine-tuning, or transfer learning. Recent works tend to employ larger models (*e.g.*, ViT [7] and Swin Transformer V2 [24]) and train them on larger datasets (*e.g.*, ImageNet-22K and JFT) in pursuit of better performance on ImageNet-1K. Both in the domains of NLP and computer vision, these large models [4, 24, 30, 9, 40] achieve enormous performance improvements compared to the small-scale models and provide guidance for downstream tasks. Some additional works attempt to explore how to efficiently fine-tune the pre-trained models [8, 41] on the target tasks. For instance, given a target task, SpotTune [8] investigates which layers need to be fine-tuned. Touvron *et al.* [34] find that fine-tuning the weights of the attention layers and freezing weights of the other parts is sufficient to adapt the vision transformers to other downstream tasks. Some works propose to insert adapters into the network to fine-tune in a parameter-efficient way. These adapters can be a small nonlinear network [14, 2], a hypernetwork that generates model weights [29], or a compactor [28] which performs a low-rank decomposition to reduce the parameters. Besides, some efforts have also been made to insert a new module into the backbone [15] or only update the bias term [1]. More recently, VPT [21] proposes to only insert a small number of trainable parameters (prompt) while freezing the backbone, which achieves significant performance improvement compared with the fully fine-tuning. Different from all above works, we propose to use the linear projection of feature and design a LIFTs-ADA module to perform the linear projection for feature, which is simple but effective and can be easily extended to a wide variety of networks.

2.3 Model Re-parameterization

Model re-parameterization has been a common practice to improve the inference efficiency. One of the representative technique is batch normalization folding used in the model compression algorithms [20]. The parameters introduced by the batch normalization layers [19] are fused into the convolution layers which are usually implemented before them. This technique has been further utilized to merge different branches of networks into a single branch inference model [5]. We deploy a similar strategy such that during the inference phase, the parameters introduced during the training phase are merged into the linear layer defined in the baseline model. In this manner, LIFTs achieves state-of-the-art accuracy without extra parameters and computation overhead during the inference stage.

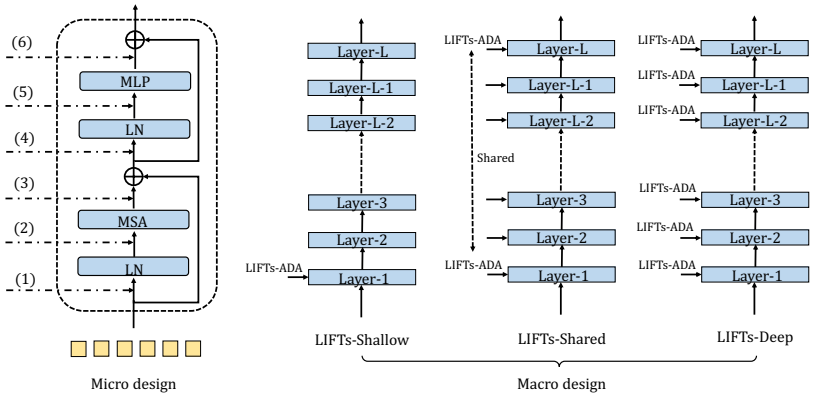


Figure 2: Our proposed LIFTs pipelines. The left figure shows the micro design. The three ones on the right show the macro design: LIFTs-Shallow, LIFTs-Shared and LIFTs-Deep.

3 Approach

In this section, we first give a recap of the preliminaries on vision transformers and a very recent work on parameter-efficient fine-tuning in Sec. 3.1. From there, we point out the two main drawbacks existing in the current proxy. We then investigate the feature scalability at the output of different components of the vision transformer backbones in Sec. 3.2. In the same section, we present how the linear transformations are added during fine-tuning. We also give a detailed analysis of the computational cost of the proposed proxy and explain in detail how LIFTs-ADA achieve zero parameter and computational cost overhead in Sec. 3.3.

3.1 Preliminaries

Transformers. In a conventional vision transformer model (ViT) [7], an RGB image $I \in \mathbb{R}^{3\times H\times W}$ is first divided into $N \times N$ non-overlapping patches. Those image patches are fed into an embedded layer where position embeddings are appended to each patch. The generated patch embeddings, also called tokens, $X_0 = \{x_0^j\}$, where $j \in [0, N^2 - 1]$ represents the j-th token. For the i-th layer where $i \in \{1, 2, ..., L\}$ for L-layer ViT, the feature tokens $X_i = \{x_i^j\}$ can be obtained through the former i layer. ViT also defines a class token x_i^{cls} that encodes the category information of the image and is used to interact with other tokens. For the i-th layer, the output is as follows:

$$\begin{cases} Z_i = X_{i-1} + \text{MSA}(\text{LN}(X_{i-1})), \\ X_i = Z_i + \text{MLP}(\text{LN}(Z_i)), \end{cases} \tag{1}$$

where MSA and LN are the multi-head self-attention and LayerNorm, respectively. For simplicity, we omit the position encoding in the first layer.

VPT. VPT [21] aims to perform fine-tuning with as few parameters as possible. Take VPT-Deep as an example, the prompts are used as extra tokens appended to the image inputs token in each layer. When inserting K prompts in the i-layer, the prompt are $P_i = \{t_i^k\}$ where $k \in [0, K-1]$. The output of the i-th layer can be calculated as:

$$\begin{cases} Z_i = X_{i-1} + \text{MSA}(\text{LN}([X_{i-1}; P_{i-1}])), \\ X_i = Z_i + \text{MLP}(\text{LN}(Z_i)). \end{cases} \tag{2}$$

During fine-tuning phase, only the parameters of P are updated. The rest of the parameters in the backbone are kept frozen.

3.2 Linear Feature Scalability Adapter (LIFTs-ADA)

In this section, we present how the proxy of our proposed Linear Feature Scalability Adapter (LIFTs-ADA) is motivated and designed. We start by conducting a set of experiments to systematically

examine the feature scalability at different positions of the network. The definition of feature scalability is formally defined as:

Definition 1 *The feature scalability describes how good the performance of fine-tuning could be achieved when performing some transformations on this feature while keeping the parameters frozen.*

We first design LIFTs-ADA and then investigate feature scalability on two levels: block level (Micro design) and architecture level (Macro design). For Micro design, we investigate how transforming the token features of each operator would impact the overall accuracy of the model. After fixing the position of token transformation, we investigate the way of transformation between different blocks.

The design of LIFTs-ADA. Combining the micro design and macro design guidance, we propose an extremely simple yet effective fine-tuning proxy. Specifically, given the feature $x \in \mathbb{R}^{d \times N^2}$ where d is the feature dimension, the output y through an operation can be formulated as $y = f(x)$, where f represents this operation function. As shown in the micro design in Figure 2, the choice of f might be LN, MSA, Residual, and MLP.

Now we consider a projection of the input feature x using transformation function g, we can get a new feature $g(x)$. If we use $g(x)$ instead of x as the input to the next operation of the network (*i.e.*, $y = f(g(x))$) and use it for fine-tuning to get good performance, we claim that this position has strong feature scalability. Here, we consider that g is a linear function for the simplicity of the linear function, and the properties of the linear function make it possible to be merged with other operations of the network in the inference stage.

Micro token feature transformation design. We try to find a micro location with stronger linear feature scalability. Depending on the insertion location, we show six situations: (1) to (6) in Figure 2. Note that we are first able to exclude (1) and (4) because the output of LN operation is invariant when a linear transformation is performed, *i.e.*, $\text{LN}(g(x)) = \text{LN}(x)$. If the linear transformation is still inserted, then g will not be learned because the parameters of the other parts in the network are frozen. In order to find which position is more feature scalable, the remaining four cases are combined to investigate their performance. The results in Sec. 4.4 show that (3)+(6) achieves better feature scalability.

Macro token feature transformation design. Are bottom blocks more important for the feature adaptation to achieve a better fine-tuned accuracy? or it should be the other way round? For either way, there are some supportive reasoning. The bottom blocks' features are capturing local level semantics which are similar across different vision datasets but essential for high quality representation learning. We try to explore this empirically by designing three set of experiments: LIFTs-Shallow, LIFTs-Shared and LIFTs-Deep. For LIFTs-Shallow setting, we only fine-tune the bottom blocks. For LIFTs-Deep, we fine-tune both the bottom and top blocks where the LIFTs-Shared setting fine-tune all blocks but apply a shared transformation. Specifically, the formulation is as follows.

$$\begin{cases} Z_i = X_{i-1} + A_i^1 \text{MSA}(\text{LN}(X_{i-1})) + b_i^1, \\ X_i = Z_i + A_i^2 \text{MLP}(\text{LN}(Z_i)) + b_i^2, \end{cases} \tag{3}$$

where A_i^1 and b_i^1 represents the linear projection for the multi-head attention module in the i-th layer and A_i^2 and b_i^2 are the linear project for the MLP.

For LIFTs-Shallow, LIFTs-ADA is only inserted into the first layer of the network and other layers remain unchanged. That is to say, Eq. (3) is only applicable when $i = 1$, and in other layers, Eq. (1) is employed. In LIFTs-Shared, LIFTs-ADA is inserted into all layers of the network, but A^1, A^2, b^1, b^2 are shared across all layers, *i.e.*, $A_i^1 = A_{i+1}^1 = ... = A_L^1 = A^1$, *etc*. In LIFTs-Deep, LIFTs-ADA is inserted into all layers of the network, but are not shared. It strictly conforms to Eq. (3). The code of LIFTs is listed in Alg. 1. The difference between the three settings is illustrated in Figure 2. The results are shown in Table 6. We empirically show that both the bottom blocks and top blocks' features are important for achieving state-of-the-art accuracy.

Zero-overhead inference with Re-parameterization. Since LIFTs-ADA is a completely linear transformation, we can re-parameterize it following the recent work [6]. The linear coefficients A

and b in Eq. (3) can be absorbed into the linear layers in the MSA and MLP operations. Consider a linear layer in MSA, which maps the input x to y through $y = A^l x + b^l$. After that, y is outputted to Eq. (3). Therefore, the final output is formulated as out $= A^1(A^l x + b^l) + b^1$. Here we omit i for the sake of simplifying presentation. Since A^l and b^l are frozen and A^1 and b^1 are learned when fine-tuning, the A^1 and b^1 can be merged into the original parameter space (A^l and b^l) in the inference stage. The operation of MLP is similar. From this perspective, our LIFTs-ADA makes it possible to perform downstream tasks without adding any extra parameters and computational costs. We also perform this operation in our experiments and find it obtains sub-optimal performance. Intuitively, modifying the feature distribution is equivalent to modifying the distribution of the weights first as $T(WX^T) = T(W)X^T$ where T denotes the linear transformation.

3.3 Complexity Analyses

We also compare the complexity of Adapter, VPT and our LIFTs-ADA. Take a ViT as an example, the dimension and number of the tokens are d and N^2, respectively. Assuming that Adapter projects features from d-dim to d'-dim, VPT inserts n prompts in each layer but LIFTs-ADA only performs scaling and shifting for the MSA and MLP modules. When the total number of layers is L, the complexity of VPT and LIFTs-ADA is shown in Table 1. As we can see, LIFTs-ADA has relatively fewer parameters and FLOPs in the training stage, which makes fine-tuning more efficient. It is worth noting that although the fine-tuning parameter of VPT is small, FLOPs grow quadratically with the number of inserted prompts (*e.g.*, $O(n^2)$), which brings a greater computational cost both during the training and inference stages. As for LIFTs-ADA, in the inference stage, borrowing the model re-parameterization strategy, the extra parameters and FLOPs of LIFTs-ADA are zero. However, the complexity of VPT remains the same compared to the training, which reflects the strengths of our approach.

Algorithm 1 Code of LIFTs-ADA in a PyTorch-like style.

```
# args: configurations
# dim: the dimension of the tokens

import torch
import torch.nn.functional as F

class LIFTs-ADA(nn.Module):
    def __init__(args):
        self.scale = nn.Parameter(torch.randn(
            dim))
        self.shift = nn.Parameter(torch.randn(
            dim))

    def forward(x):
        x = self.scale * x + self.shift

    return x
```

Method	Adapter	VPT-Shallow	VPT-Deep	LIFTs-Shallow	LIFTs-Shared	LIFTs-Deep
#Params.	$2Ldd'$	nd	nLd	$4d$	$4d$	$4Ld$
#FLOPs	$2N^2Ldd'$	$2n(2N^2+n)d$	$2n(2N^2+n)Ld$	$4N^2d$	$4N^2d$	$4N^2Ld$

Table 1: The complexity of Adapter [14], VPT [21] and our LIFTs-ADA.

4 Experiments

To validate the effectiveness of our proposed approach, we have conducted extensive experiments. In this section, we first introduce the experimental settings. Then, we show the experiments for the comparisons with baseline proxies on image classification. We then show the ablations on the impacts of different module designs. After that, we show that LIFTs do learned better representations compared to naive linear probing fine-tuning. To further verify the power of LIFTs, We also validate the effectiveness of our method on the object detection and semantic segmentation tasks. Due to the space limitation, the results are put in the supplementary material.

4.1 Experimental Settings

Models. For a fair comparison, we follow VPT [21] and mainly select ViT-B/16 [7] pre-trained on ImageNet-21K as the initialization for fine-tuning. In addition, we also generalize our method to backbones of different model families, including the recent Swin Transformer [24] (Swin-B),

ConvNext-B [25] and AS-MLP-B [23], the latter two belong to CNN-based architecture and MLP-based architecture respectively.

Baselines. We first compare our method with the two basic fine-tuning methods: i) full fine-tuning, where all parameters of the models are updated at fine-tuning; ii) linear probing, where only parameters of classification head (a MLP layer) are updated. We also compare our method with recent parameter-efficient fine-tuning methods: iii) adapter [14], where a new MLP with up-projection, non-linear function and down-projection is inserted into the transformer and the parameters of these new modules are updated; iv) bias [38], where the bias parameters are fine-tuned; v) VPT [21], where the prompts are inserted into transformers as tokens and they are updated at fine-tuning.

Implementation details. For the FGVC datasets, we follow the same data augmentation settings adopted in VPT [21]. Specifically, we process the image with randomly resize crop to 224×224 and random horizontal flip for data augmentation. For other datasets, we follow the original settings of each model for fine-tuning on ImageNet-1K, COCO and ADE20K, where stronger data augmentation strategies are adopted. We employ the AdamW [27] optimizer to fine-tune models for 100 epochs for FGVC and CIFAR-100, and 30 epochs for ImageNet-1K. The cosine decay strategy is adopted for the learning rate schedule, and the first 10 epochs use linear warm-up.

Method \ Dataset	CUB-200-2011	NABirds	Oxford Flowers	Stanford Dogs	Stanford Cars	Mean
Full fine-tuning	87.3	82.7	98.8	89.4	84.5	88.54
Linear probe	85.3	75.9	97.9	86.2	51.3	79.32
Bias	88.4	84.2	98.8	91.2	79.4	88.41
Adapter	87.1	84.3	98.5	89.8	68.6	85.67
VPT-Deep	88.5	84.2	99.0	90.2	83.6	89.11
Tuned / Total (%)	0.29	1.02	0.14	1.17	2.27	0.98
FLOPs	96.37 M	517.94 M	47.73 M	1128.09 M	2625.15 M	883.06 M
LIFTs-Deep (ours)	88.9	83.1	99.0	89.4	83.2	88.88
Tuned / Total (%)	0.215	0.086	0.144	0.128	0.212	0.157
FLOPs	9.59 M	9.48 M	9.52 M	9.53 M	9.59 M	9.54 M

Table 2: Performance comparisons on five FGVC datasets.

4.2 Performance Comparisons on Image Classification

Datasets. We mainly conduct our experiments on a series of image classification datasets as follows: i) FGVC, which contains five fine-grained visual classification tasks; ii) CIFAR-100 and ImageNet-1K, which are the general object classification datasets. Details of the dataset can be found in the supplementary material.

Results. We compare the performance of our LIFTs-Deep, VPT-Deep and other baselines on the FGVC datasets and the results are shown in Table 2. We can see that, with less parameter and computational overhead, LIFTs-Deep proxy trained models achieves comparable or even better results than VPT-Deep. We argue this handy results are mainly owning to the linear projection applied on the feature tokens for parameter-efficient fine-tuning. From the 'prompt length' metrics, it is observed that VPT-Deep proxy needs to design different number of prompts for each task, making it hard to be directly used. Differently, LIFTs only needs to apply a scaling factor and a shift factor on the output of self-attention module and MLP module in each block, respectively. This is much more concise and efficient. It is worth noting that Table 2 only shows the extra parameters and FLOPs in the training stage. These will be reduced to zero with model re-parameterization in the inference stage. As mentioned before, VPT might not be friendly for those class-token-free models, *e.g.*, Swin Transformer. The strategy of add extra prompts for self-attention in VPT is equivalent to add additional class token to fine-tune for these models. To verify the effectiveness of LIFTs on a wide variety of models, we conduct experiments on Swin Transformer, ConvNext, and AS-MLP, which

[1]Since VPT-Deep is not trained in the full CIFAR-100 dataset and does not report performance on ImageNet-1K, we re-implement VPT-Deep with Swin-B. See the supplementary material for details. For ConvNext-B and AS-MLP-B, the core operations are convolution and MLP and they do not need self-attention, we try to insert prompt and perform self-attention with features, however, we do not obtain good results.

Method	ViT-B/16 [7]		Swin-B [24]		ConvNext-B [25]		AS-MLP-B [23]
	CIFAR-100	ImageNet-1K	CIFAR-100	ImageNet-1K	CIFAR-100	ImageNet-1K	CIFAR-100
Full fine-tuning	**93.69**	**83.58**	**93.24**	**85.20**	**93.51**	**85.80**	**89.98**
Linear probing	87.28	80.31	89.42	83.25	90.30	84.05	79.05
Adapter [14]	92.42	82.66	92.08	83.82	92.76	84.53	84.10
Bias [38]	92.21	82.75	92.19	83.92	92.80	84.79	84.59
VPT-Shallow [21]	91.28	81.43	90.40	82.71	-	-	-
VPT-Deep[1] [21]	92.43	82.69	91.05	83.44	-	-	-
LIFTs-Deep (ours)	92.76	82.82	92.47	84.24	92.91	85.01	86.11

Table 3: Performance comparisons on CIFAR-100 and ImageNet-1K with various models. The bold font shows the best accuracy of all methods and the underline font shows the second best accuracy.

belongs to three different types of architectures (Transformers, CNNs, and MLPs). The experimental results are shown in Table 3. we perform the full fine-tuning, linear probe, VPT-Deep [21], and LIFTs-Deep on the CIFAR-100 and ImageNet-1K datasets, respectively. LIFTs-Deep outperforms VPT-Deep on the CIFAR-100 and ImageNet-1K datasets with Swin-B architecture, and our results are also close to the full fine-tuning on a challenging dataset like ImageNet-1K, which validates the effectiveness and linear feature scaling to a variety of models.

Computational cost. To validate the efficiency of our method, we show the computational cost of LIFTs-Deep in Table 4. For the training stage, we employ a batch size of 16 and mixed precision training. For the inference stage, the batch size is 1. The running time in Table 4 is measured in a single GeForce RTX 2080Ti GPU with 11G memory in total. We can see that LIFTs-Deep has faster training and inference time, and less memory consumption than VPT-Deep, where we test the computational cost with 10 prompts following [21]. When adding the number of prompts, the time cost and memory will be larger as shown in Table 12 of VPT [21]. However, LIFTs-Deep achieves zero-overhead inference with re-parameterization, which is more advantageous.

Swin-B (88 M)	Train		Test	
	Time (ms)	Memory (GB)	Time (ms)	Memory (GB)
Full fine-tuning	273	4.90	52	0.96
Linear probe	149	1.91	51	0.91
Adapter [14]	165	3.59	71	0.99
Bias [38]	175	3.54	51	0.95
VPT-Shallow [21]	161	3.83	69	1.02
VPT-Deep [21]	183	4.03	75	1.07
LIFTs-Deep (ours)	159	3.64	52	0.96

Table 4: Computational cost of different methods.

4.3 Performance Comparison on Robustness and OOD

Datasets. We also conduct experiments to analyze the robustness and Out-Of-Distribution (OOD) ability of our LIFTs with the following datasets: ImageNet-A, ImageNet-R and ImageNet-C. Please refer to the supplementary material for their details.

Results. We perform robustness and OOD evaluation on the ImageNet-A, ImageNet-R and ImageNet-C datasets with the fine-tuned models on ImageNet-1K. All experimental results are listed in Table 5. We have two findings from this table: i) our LIFTs obtains better performance than VPT on both datasets, which shows our fine-tuning method has stronger robustness and out-of-distribution generalization; ii) although LIFTs has lower accuracy than full fine-tuning on ImageNet-1K, the performance on ImageNet-A, ImageNet-R and ImageNet-C is better. However, the performance between ImageNet-1K and ImageNet-A (ImageNet-C) is not absolutely positive relevant. We believe such improvements in robustness and OOD datasets might come from the fact that LIFTs freeze most of the pre-trained parameters and thus

Method \ Dataset	IN-1K	IN-A	IN-R	IN-C
Full fine-tuning	83.58	34.49	51.29	46.47
Linear probing	80.31	29.43	50.83	49.04
VPT-Deep [21]	82.69	44.03	53.41	42.05
LIFTs (ours)	82.82	45.88	56.77	41.47

Table 5: Performance comparisons on robustness and out-of-distribution datasets. 'IN' means ImageNet.

Comb.	Acc. (%)	Comb.	Acc. (%)	Comb.	Acc. (%)
(2)	92.24	(2)+(3)	92.34	(3)+(6)	92.47
(3)	92.34	(2)+(5)	92.43	(5)+(6)	92.38
(5)	92.35	(2)+(6)	92.32	(2)+(3)+(6)	92.45
(6)	92.28	(3)+(5)	92.41	(3)+(5)+(6)	92.47

(a) The impacts of the different insertion locations. We use 'Acc.' for accuracy and 'Comb.' for combination.

Method	Acc. (%)	Tuned / Total (%)
Full fine-tuning	93.24	100
Linear probe	89.42	0.087
LIFTs-Shallow	90.62	0.088
LIFTs-Shared	91.15	0.096
LIFTs-Deep	92.47	0.142

(b) The impacts of different variants.

Table 6: The Impacts of Different Designs.

maximally preserve the knowledge learned on the large dataset for pre-training and thus maintain a better generalization capability.

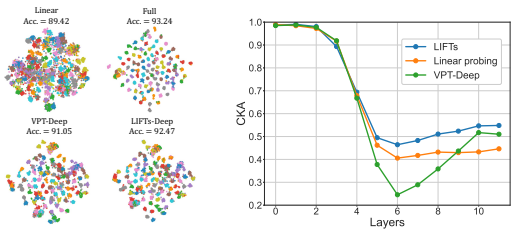
4.4 The Impacts of Different Designs

As described in Sec. 3.2, we first explore the impact of the different insertion locations. We combine the different locations in Figure 2 for evaluation and the results are shown in Table 6a. All experiments are conducted on CIFAR-100. We have two findings as follows: i) Combining two insert positions have better performance than only one position. However, the performance at the combination of three positions has reached saturation and can hardly be further improved; ii) hybrid module insertion in MSA and MLP has better performance than single insertion of two projection functions in one module, which implies better linear feature scalability with mixed layer insertion in MSA and MLP.

Furthermore, we compare the performance of LIFTs-Shallow, LIFTs-Share, and LIFTs-Deep in Table 6b with the Swin-B architecture. We use the obtained insertion configuration ((3)+(6)) from Table 6a. We can see that all methods outperform the linear probe, which shows the effectiveness of our method.

4.5 Visualization

We also visualize the feature distribution of CIFAR-100 with different methods in Figure 3a. Our LIFT-Deep achieves better feature clustering results compared to the linear probe and VPT-Deep, which attributes to our LIFTs-ADA layer. Besides, to further understand LIFTs, we also visualize the feature similarity between full fine-tuning and LIFTs, full fine-tuning and Linear probing, full fine-tuning and VPT-Deep. The results are shown in Figure 3b. In the last layer, LIFTs has the most similar feature to full fine-tuning and the accuracy is also the closest.



(a) (b)

Figure 3: (a) shows the T-SNE visualization of different methods, including linear probe, full fine-tuning, VPT-Deep, and our LIFTs-Deep; (b) shows the feature similarities between full fine-tuning and different methods in different layers.

5 Conclusion

In this paper, we focus on the parameter-efficient fine-tuning task and surprisingly find that a simple linear adapter is capable of such a task. Different from previous full fine-tuning and other parameter-efficient works, where some complex non-linear operation is introduced or a prompt is inserted into the input of the network, we explore linear feature scalability of vision transformer and design LIFTs-ADA. Furthermore, our linear adapter can be merged into the network during the inference stage with re-parameterization, which allows the fine-tuned model to be applied to downstream tasks without adding any additional parameters and FLOPs compared to the pre-trained model. We conduct the experiments on various datasets and models, which shows comparable or even better performance compared with VPT. Such a method effectively saves computational resources and reduces carbon emissions. For future work, we will explore the effectiveness of LIFTs-ADA in NLP and more downstream tasks.

References

[1] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. Tinytl: Reduce memory, not parameters for efficient on-device learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11285–11297. Curran Associates, Inc., 2020.

[2] Shoufa Chen, Chongjian Ge, Zhan Tong, Jiangliu Wang, Yibing Song, Jue Wang, and Ping Luo. Adaptformer: Adapting vision transformers for scalable visual recognition. *arXiv preprint arXiv:2205.13535*, 2022.

[3] Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. *Advances in Neural Information Processing Systems*, 34:3965–3977, 2021.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13733–13742, 2021.

[6] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. *arXiv preprint arXiv:2101.03697*, 2021.

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

[8] Yunhui Guo, Honghui Shi, Abhishek Kumar, Kristen Grauman, Tajana Rosing, and Rogerio Feris. Spottune: transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4805–4814, 2019.

[9] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. *arXiv preprint arXiv:2111.06377*, 2021.

[10] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[11] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4918–4927, 2019.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[13] Xuehai He, Chunyuan Li, Pengchuan Zhang, Jianwei Yang, and Xin Eric Wang. Parameter-efficient fine-tuning for vision transformers. *arXiv preprint arXiv:2203.16329*, 2022.

[14] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[15] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR, 09–15 Jun 2019.

[16] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, To Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[17] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[18] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[20] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[21] Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning. *arXiv preprint arXiv:2203.12119*, 2022.

[22] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

[23] Dongze Lian, Zehao Yu, Xing Sun, and Shenghua Gao. As-mlp: An axial shifted mlp architecture for vision. *ICLR*, 2022.

[24] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2021.

[25] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[26] Ze Liu, Jia Ning, Yue Cao, Yixuan Wei, Zheng Zhang, Stephen Lin, and Han Hu. Video swin transformer. *arXiv preprint arXiv:2106.13230*, 2021.

[27] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

[28] Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. *arXiv preprint arXiv:2106.04647*, 2021.

[29] Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. *arXiv preprint arXiv:2106.04489*, 2021.

[30] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

[32] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning (ICML)*, pages 6105–6114, 2019.

[33] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning (ICML)*, 2021.

[34] Hugo Touvron, Matthieu Cord, Alaaeldin El-Nouby, Jakob Verbeek, and Hervé Jégou. Three things everyone should know about vision transformers. *arXiv preprint arXiv:2203.09795*, 2022.

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[36] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in Neural Information Processing Systems*, 34, 2021.

[37] Saining Xie, Ross Girshick, Piotr Dollar, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[38] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*, 2021.

[39] Daquan Zhou, Bingyi Kang, Xiaojie Jin, Linjie Yang, Xiaochen Lian, Zihang Jiang, Qibin Hou, and Jiashi Feng. DeepViT: Towards deeper vision transformer. *arXiv preprint arXiv:2103.11886*,

2021.

[40] Daquan Zhou, Zhiding Yu, Enze Xie, Chaowei Xiao, Anima Anandkumar, Jiashi Feng, and Jose M Alvarez. Understanding the robustness in vision transformers. *arXiv preprint arXiv:2204.12451*, 2022.

[41] Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. Learning to prompt for vision-language models. *arXiv preprint arXiv:2109.01134*, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [No]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]